

BugSecure: Secure Crowdsourced Bug Bounty Platform for Vulnerability Management

Dr. Monali Tingane

School of Computing
MIT ADT UNIVERSITY

Pune, India

monali.tingane@mituniversity.edu.in

Naman Babbar

School of Computing
MIT ADT UNIVERSITY

Pune, India

namanbabbar37@gmail.com

Goutam Kumar

School of Computing
MIT ADT UNIVERSITY

Pune, India

goutamp241@gmail.com

Vasu Kamra

School of Computing
MIT ADT UNIVERSITY

Pune, India

vasukamra007@gmail.com

Abstract—This paper proposes a concept of a "Secure Crowdsourced Bug Bounty Platform" that is intended to help connect organizations with ethical hackers, thereby enabling vulnerability management. This platform is intended to offer a framework with secure authentication, role-based access control, encryption, scalability, payment gateways, etc., to help small and medium enterprises carry out cost-effective vulnerability reporting.

Index Terms—Crowdsourcing, Cybersecurity, Bug Bounty, Ethical Hacking, Vulnerability Management, Secure Platform, Data Protection, SMEs.

I. INTRODUCTION

With software becoming ubiquitous in critical infrastructures, healthcare, finance, and day-to-day devices, software security is considered a top priority [8]. Software security issues have the potential to result in data breaches, financial loss, and even threats to human lives [67]. Conventional security methods, such as relying only on internal testing and periodic auditing, have failed to address software security issues [62].

Bug bounty programs have been found to be highly effective in engaging external security experts to help fix software security issues [28]. Platforms such as HackerOne and Bugcrowd have enabled thousands of critical software security issues to be discovered in prominent technology companies [73]. However, there is a lack of integration between bug bounty programs and internal software development processes, which is affecting their efficient use [43].

BugSecure solves these problems by offering a unified framework that: (1) integrates bug bounty programs with continuous integration and deployment processes; (2) uses automated static and dynamic analysis tools in conjunction with human-driven testing; (3) offers a standard approach for vulnerability reporting and remediation; and (4) offers metrics and dashboards for the assessment of the security posture [50]. The contributions of this paper are as follows:

- We describe our BugSecure architecture and some of its main components.
- We explain our design for our integrated triage and validation pipeline.
- We present our results from a multi-project evaluation.
- We offer a brief discussion on some of our limitations and future work.

Today, organizations must contend with an ever-expanding attack surface, from web applications and APIs, mobile applications, cloud infrastructure, and IoT devices, which demand constant security attention [67]. Organizations' internal security resources are often stretched thin, dealing with a large volume of potential vulnerabilities [63]. Bug bounty programs provide a way for organizations to leverage a global community of security researchers, but without proper tooling, the influx of reports can create a bottleneck, slowing down remediation efforts [43]. Reports may be duplicated across platforms or may be mismanaged, leading to analyst fatigue. BugSecure is intended to alleviate this by providing an integrated, measurable solution for managing vulnerability reports from human and automated sources.

Moreover, with the advent of DevOps and continuous deployment, time-to-release is much shorter, making it hard to keep up with traditional security checks [61]. Thus, security teams have to use tools that integrate well with their processes while still allowing them to have visibility into externally reported bugs. BugSecure is created to solve this problem by providing a unified, measurable method of dealing with vulnerability reports, whether they come from humans or automated tools. Our framework doesn't replace bug bounty platforms or internal tools, but rather brings them together, providing uniform processes for triage, validation, and remediation, so that security teams can grow their processes without increasing their headcount.

The rest of this paper is structured as follows. Section II discusses the existing research in the area of bug bounty

programs, vulnerability disclosures, the secure development lifecycle, and security metrics. Section III offers background information regarding the lifecycle of vulnerabilities, the threat model, and the notation. Section IV discusses the methodology that has been adopted for this research. Section V discusses the system design for the proposed system. Section VI offers information regarding the implementation of the proposed system. Section VII discusses the evaluation of the proposed system. Section VIII offers the implications and the limitations of the proposed system. Section IX concludes the paper.

A. Motivation and Scope

The scope of this document is limited to vulnerability management workflow involving both external bug bounty data as well as internal tooling, while avoiding other topics such as vulnerability research methodologies, exploit development, and offensive security practices in general. This document is targeted at security engineers, program managers, and development leads who are responsible for triaging, validating, and remediating vulnerabilities in their production or pre-production environments. It is assumed that an organization has a bug bounty program in place, or at least has a plan to implement one, as well as an issue tracker system, which BugSecure integrates with.

II. LITERATURE SURVEY

Existing literature on crowdsourcing and cybersecurity emphasizes platform security, coordination of participants, and secure communication, which are essential aspects of building a sustainable bug bounty program [1], [2]. However, there is a research gap where current platforms may not be appropriate for smaller companies due to their complexity. This paper proposes a new platform called "Secure Crowdsourced Bug Bounty Platform," which includes secure authentication, transparent rewards, and a vulnerability management process [3]–[5].

A. Bug Bounty Programs and Crowdsourced Security

Bug bounty programs motivate external researchers to identify and disclose vulnerabilities in return for financial rewards and recognition [11], [28]. These programs have gained popularity in the past decade, with large tech companies and government organizations adopting them as an additional means of testing their software. In a related study, Cavusoglu et al. [16] discussed the economics of vulnerability markets and how the right bug bounty programs can align the interests of vendors and external researchers. This will lead to a more efficient discovery of vulnerabilities. In a different study, Zhao et al. [73] conducted an empirical analysis of crowdsourced security platforms and found factors that could be used to predict the success of bug bounty programs. These factors include the amount of the reward, the scope of the program, the response time, and the maturity of the bug bounty program. This indicates that the right bug bounty programs will attract a large number of external researchers and result in a higher quality of reports. Mailloux and Stamp [43] have researched

challenges related to integration, especially while integrating bug bounty programs. According to their study, many companies do not have a standard process to deal with bug bounty programs. Thompson and Ramesh [64] have researched how to measure bug bounty program effectiveness, such as time to triage, time to validate, and time to remediate, which is used while evaluating bug bounty programs. Lasseter et al. [39] have researched how diversity affects bug bounty researchers, specifically how it affects vulnerability coverage. Government and public-sector bug bounty program adoption is researched by Riggs and Gordon [55] and Chen and Wang [20].

B. Vulnerability Disclosure and Coordination

Responsible disclosure balances the need to fix vulnerabilities with the potential impact of premature disclosure that may cause undue damage [11], [37]. In a study by Kannan and Telang [37], they questioned whether a market for vulnerabilities leads to optimal results and proposed alternatives. Finifter et al. [27] provided data and evidence about vulnerability disclosure and reward schemes in vulnerability rewards programs and showed that faster vendor response correlates with increased researcher engagement. Zhang and Wang [71] studied the effect of vulnerability disclosure policies on vendor and researcher behavior and emphasized the need for clear policies. Householder et al. [34] provided evidence about vulnerability disclosure and the role that CERT/CC plays in facilitating communication between vulnerability researchers and vendors in vulnerability disclosure, including guidelines for disclosure timing. Van Egas and Rueda [66] provided a survey about vulnerability disclosure policy adoption and best practices and recommended that organizations should have explicit vulnerability disclosure policies in place to reduce legal ambiguity. Bell and Owens [13] discussed legal and policy issues in vulnerability disclosure, which is relevant to our study in designing our vulnerability disclosure and remediation process.

C. Secure Development Lifecycle and Automation

The importance of integrating security into the software development lifecycle (SDL) was advocated by Howard and Lipner [35] and OWASP [51]. The Microsoft Security Development Lifecycle and similar approaches also emphasize security requirements, security design, and security testing. Chess and West [21] and McGraw [44] also introduced basic frameworks for building security into software, which include static analysis. Automated static analysis for vulnerability detection has been extensively studied. Livshits and Lam [42] showed the effectiveness of finding security vulnerabilities in Java applications using static analysis. Johnson et al. [36] also studied the reasons for the lack of adoption of automated static analysis by software developers. Allamanis et al. [6] and Li et al. [41] also applied machine learning for vulnerability detection. VulDeePecker is a deep learning approach for vulnerability detection in source code. Avgerinos et al. [12] and Cha et al. [18] also improved symbolic execution and fuzzing for bug detection. Böhme et al. [14] also reviewed

TABLE I: Table I. Literature Review

Title	Author(s)	Result
A Systematic Literature Review of Crowdsourcing-Based Research in Information Security	Karachiwalla, R., & Pinkow, F.	This article highlights the major design factors in the development of crowdsourcing-based information security projects.
Cyber Risk Assessment in Small and Medium-Sized Enterprises (SMEs)	Sukumar, A	This article highlights the challenges faced by SMEs in the field of cybersecurity.
Crowdsourcing for Security in the Age of Hybrid Threats	Dobák, I.	This article highlights the benefits of crowdsourcing in the field of cybersecurity.
The MTNK Platform for Making Cybersecurity Accessible to SMEs	Igini, F	This article highlights the benefits of the collaborative platform in the field of cybersecurity.
A Survey of Bug Bounty Programs in Strengthening Cybersecurity	Arshad, J.	This article highlights the bug bounty programs and their benefits in the field of cybersecurity.

fuzzing approaches. Nguyen et al. [47] reviewed deep learning approaches for vulnerability detection. These automated approaches can be combined with human-driven approaches for bug bounty, as BugSecure also includes.

D. Security Metrics and Risk Management

Schneier [59] and Gordon/Loeb [30] have discussed the concept of risk management in cybersecurity, which also encompasses cost-benefit analysis for security investments. Herley [33] has discussed the problems with current security metrics and suggested more effective ways to measure them in relation to actual reduced risk. Anton et al. [9] and Savola [58] have discussed security metrics for software and system security, which include vulnerability management and remediation. Recent studies by Verizon [67] and Symantec [63] present annual reports based on data breaches and vulnerability statistics. Recent studies on using natural language processing for automated triage of vulnerability reports [68] and prioritization of software vulnerabilities [38] are also relevant to our work. Legal and compliance aspects of bug bounty programs have also been discussed in recent studies by Brown/Davis [15] and Rodriguez/Martinez [56]. The psychological aspects of researchers' participation have also been studied by Rossi/Martino [57], whereas recent studies by Singh/Kumar [61] and Williams/Meneely [?]enchmarking of vulnerability detection tools [72] and reproducibility in vulnerability research [19] are additional topics that may be relevant in the context of assessing our framework. Economic analysis of bug bounty programs [40] and legal aspects [15] are relevant to the design of our rewards and program structure. Meta-analyses of crowdsourced security [46] and reassessment of the markets for vulnerability [7] are also relevant in this context. Education and training for security researchers [65], discussed by Thomas and Bailey

E. Gap and Our Position

Despite the increasing popularity of bug bounty programs and the availability of various vulnerability management tools,

there is still a disconnect between the output of bug bounty platforms (and vulnerability management tools) and the required input to various development processes. Most organizations use manual methods such as copy-paste between tools, ad hoc severity levels, and spreadsheets to manage vulnerability data. BugSecure bridges this disconnect by providing a customized integration layer with flexible triage, validation, and remediation processes, and metrics that enable operational and strategic decision-making. Our framework is agnostic to bug bounty platforms or issue trackers, providing normalized inputs and outputs to allow for changes or additions to bug bounty platforms without affecting internal triage, remediation processes, etc.

III. BACKGROUND AND PRELIMINARIES

A. Vulnerability Lifecycle

A common life cycle of a vulnerability includes discovery, reporting, triage, validation, remediation, and disclosure [17]. Discovery can happen through internal testing, automated tools, or external sources. Reporting involves submitting a description, a proof of concept, and affected assets to an organization or a bug bounty platform. Triage categorizes a reported vulnerability as a duplicate, invalid, or new valid issue. Validation checks whether a reported vulnerability is reproducible and accurately described. Remediation involves developing a patch to fix a vulnerability. Disclosure is often conducted in coordination with a researcher and the public after a patch is available. BugSecure implements this life cycle of a vulnerability, including tools to support each step, particularly focusing on triage and validation automation, while involving human experts to handle exceptions as necessary. We use definitions provided by MITRE [24], [25] to classify a vulnerability (CVE) and weakness (CWE). For severity scores, we use National Vulnerability Database [49] and CVSS [29].

B. Threat Model

We consider an organization that is either building or running a piece of software and seeks to enhance its overall

security posture through a mix of internal organizational processes and external bug bounty programs. We consider that the organization already has existing development and delivery pipelines in place and may already be using bug bounty platforms such as HackerOne or Bugcrowd, as well as bug tracking tools such as Jira or GitHub Issues. We also consider that the adversary may have differing skill sets ranging from automated tools to human researchers [60]. Our framework does not consider that all bugs can be identified through automated means. Thus, the human-in-the-loop aspect is included for both external and internal verification. We also consider that the BugSecure system is running in a trusted environment and that only authorized personnel have access to the sensitive report information. However, threat modeling for the framework itself is outside the scope of this paper but is important for production use cases.

C. Notation and Assumptions

Hereafter, we denote the set of issues by $\mathbf{I}$ and the subset of valid issues by $\mathbf{V} \subseteq \mathbf{I}$. Each issue $i \in \mathbf{I}$ has a severity level $s(i)$ and a timestamp $t(i)$. We use time-to-triage (TTT), time-to-validate (TTV), and time-to-remediate (TTR) metrics as KPIs [64]. TTT is defined as the time between report submission and initial decision based on report content (e.g., duplicate report, new report, report severity). TTV is defined as the time between initial report decision and validation decision (e.g., report is valid or not reproducible). TTR is defined as the time between report validation and fix deployment. We assume that reports are submitted in good faith and that organizations have a commitment to respond within agreed-upon SLAs; legal and policy issues have been addressed in previous work [15], [52]. We assume that it is possible to normalize severity levels across sources (e.g., bug bounty platform severity levels and CVSS).

D. Scope of Automation

Not all triage and validation steps can be or should be automated. Duplicate detection can be automated with high accuracy if thresholds for similarity are properly set. Severity normalization can be automated if source and target schema definitions are clear. Validation for whether or not a vulnerability is reproducible is typically performed by humans in a sandbox environment or through automated checks if they are available.

IV. METHODOLOGY

A. Design Goals

BugSecure was conceived with the following objectives in mind: (G1) minimize internal security teams' workload by automating triage and validation whenever feasible, allowing security analysts to focus on high-severity and edge-case reports; (G2) offer a single pane of glass for vulnerability reports from internal tools such as SAST and DAST, and external sources such as bug bounty programs; this will prevent security analysts from switching between multiple dashboards to monitor vulnerability reports; (G3) offer a unified view

for severity assessment and prioritization across all sources; and (G4) offer audit trails and metrics for compliance and improvement metrics such as TTT, TTV, TTR, and trends [50]. These objectives were shaped after a dialogue with security engineers from our partner organizations that expressed pain points with ad-hoc security workflows in place currently.

B. Data Collection and Analysis

We gathered data from the following sources: (1) anonymized historical vulnerability reports from partner organizations, used for fine-tuning deduplication thresholds and validating the effectiveness of the triage pipeline; (2) synthetic workloads created from known CVE data [49], employed for offline testing of scalability and accuracy; and (3) a controlled deployment of BugSecure in two mid-size software projects over six months, serving as the primary source of data for evaluating the tool. Quantitative measures of the effectiveness of the tool, including the number of valid vulnerabilities detected, TTT/TTV/TTR, and false positives in the triage phase, were employed in conjunction with qualitative measures of the benefits and drawbacks of the tool through surveys and interviews with security engineers and developers [23]. The analysis of the data gathered employed a mixed-methods approach.

C. Ethical Considerations

All human subject interactions were performed with IRB approval. The vulnerability data was anonymized and aggregated for publication, with no disclosure of identifiable vulnerabilities, reporters, or organizations. We adhered to the principles of responsible disclosure, which include not revealing unremediated vulnerabilities [45]. We obtained consent from the partner organizations regarding the use of anonymized metrics, with data handling compliant with the organizations' data handling policies. Our methodology was reviewed for compliance with institutional and industry standards for security research [45], [52]. We made no effort to influence participants' reporting of positive outcomes, as the survey was voluntary.

D. Stakeholder Involvement

The design goals and evaluation criteria were further refined with the help of security engineers and program managers from two partner organizations. Two sets of feedback were conducted: one for prioritizing pain points (e.g., duplicated reports, inconsistent severity, lack of visibility), and another one after a prototype demo, where we verified that the proposed architecture indeed addressed the identified pain points. This involved working with stakeholders to make sure the framework is useful, rather than just theoretically correct.

V. BUGSECURE SYSTEM DESIGN

A. Architecture Overview

BugSecure is composed of four primary layers: (1) **Ingestion** - gathering reports from bug bounty sites, internal scanners, and manual submissions; (2) **Triage & Validation** -

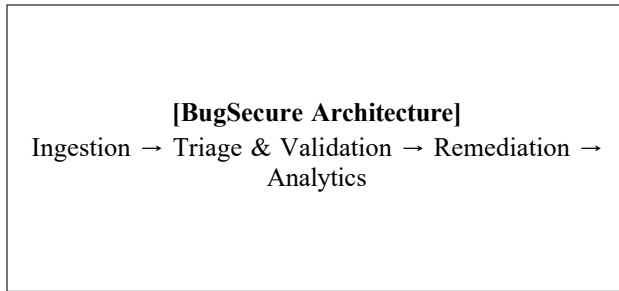


Fig. 1: BugSecure high-level architecture.

deduplication, severity scoring, and automated or manual validation; (3) **Remediation** - generation of tickets, assignment, and tracking; and (4) **Analytics & Reporting** - dashboards, trend analysis, and KPIs [50].

Figure 1 is a diagram of the process. Reports may be ingested through APIs (HackerOne, Bugcrowd, etc.) or web forms. The ingestion layer normalizes the reports into a standard data structure. The triage engine performs deduplication, severity scoring, and routing. In some cases, automated validation is performed. The issues are then routed to an issue tracker (Jira, GitHub Issues) and associated with the code and release branches. Notifications to reporters and bounty status changes occur when the fix is deployed. Analytics is performed for management dashboards and compliance reporting.

B. Ingestion Layer

The ingestion layer is responsible for normalizing data from various sources, such that all data is represented in a common format that can be processed uniformly by downstream components. Each report contains: source identifier (for example, report ID on HackerOne), title, description, severity (if available from the source), proof of concept or steps to reproduce, assets affected (for example, URLs, endpoints, components), and reporter information (anonymized or pseudonymized, depending on policy). We have support for webhook-based real-time ingestion, such that new reports from bug bounty platforms can be processed in real-time, as well as batch-based ingestion for historical data migration [10]. The schema is extensible to accommodate future sources, such as other bug bounty platforms, or internal scanner data. Normalization includes cleaning and tokenizing text data for use in deduplication and search.

C. Triage and Validation Pipeline

The four steps in the triage process are as follows: (1) Deduplication against known issues using similarity scoring for titles, descriptions, and affected assets [32]. We utilize a certain threshold for this purpose, for instance, Jaccard similarity > 0.85 . Cases that do not meet this threshold are escalated for human analysis. (2) Severity normalization across sources using CVSS [29], or a lookup table for source-specific severity levels and their normalization to a common ordinal scale. (3) Automated or human analysis based on the severity and availability of resources. This is typically performed based

on the severity and availability of resources. (4) Determination of the priority for remediation based on the severity. Our pipeline eliminates unnecessary work in the following ways: for instance, it eliminates the need for duplicate tickets for the same issue [64].

D. Remediation and Disclosure

Remediation tracking is also integrated with the existing development process. When the reported issue is validated and accepted, a new issue entry is created in the organization's existing issue tracking system (e.g., Jira) with the severity and description of the issue and a link to the original report. The development teams remediate the issue as part of their regular release cycles. When the remediation is deployed, the system may send notifications to the reporters and update the bounty status in the bug bounty platform. The disclosure of the remediation is typically controlled by the organization's policy (e.g., disclose the remediation after the patch is available, or after a grace period), and it may be automated for coordinated disclosure [34]. We support a disclosure workflow so that organizations can align with their legal and compliance requirements.

E. Analytics and Reporting

The analytics layer also aggregates data related to ingestion, triage, validation, and remediation to display analytics such as dashboards and reports. Some of the main analytics include: (1) volume of reports generated over time broken down by source and severity; (2) TTT, TTV, and TTR distributions; (3) backlog size; (4) researcher activity and satisfaction (when available through the bug bounty platform). Reports can also be exported to meet compliance needs or to satisfy management needs.

We do not store raw report content in the analytics store except to support deduplication and search; we store this information in the operational store as required, including access control. Dashboards can also display information depending on the role; e.g., security engineers can see detailed report information, while management can see aggregated information such as trends. Export formats can include CSV and PDF to meet compliance needs as well as reporting needs.

F. Security and Privacy of the Framework

BugSecure itself must be secure with respect to access control, encryption of sensitive report data, and audit logs [51]. Principle of least privilege and data encryption at rest and in transit are implemented [48]. Table II lists security controls implemented in our system.

VI. IMPLEMENTATION

A. Technology Stack

The prototype has been developed as a microservices architecture to allow independent scaling and deployment of each of the components, including ingestion, triage, remediation, and analytics. Python has been chosen as a backend language to develop the triage and validation components, as we can

TABLE II: Security Controls in BugSecure

Control	Implementation	Standard
Access control	Role-based (RBAC)	NIST RBAC
Encryption at rest	AES-256	NIST FIPS 197
Encryption in transit	TLS 1.3	RFC 8446
Audit logging	Immutable, tamper-evident	SIEM
Data retention	Configurable	GDPR/CCPA

Algorithm 1 Triage Pipeline

```

1: Input: Report  $r$  from ingestion layer
2: Normalize  $r$  (title, description, affected assets)
3: for each existing issue  $i \in \mathbf{I}$  do
4:   Compute similarity  $\sigma(r, i)$  (e.g., Jaccard on tokens)
5:   if  $\sigma(r, i) > \vartheta_{\text{dup}}$  then
6:     return duplicate of  $i$ 
7:   end if
8: end for
9: Map severity  $s(r)$  to common scale
10: Assign priority  $p(r)$  from  $s(r)$  and SLA policy
11: return new issue; enqueue for validation

```

leverage libraries such as NLP and similarity tools. On the other hand, Node.js has been chosen to develop the ingestion webhook handlers as well as the real-time notification system. PostgreSQL has been chosen as a backend database to store data, along with full-text search capabilities such as fuzzy search using the `pg_trgm` extension to allow deduplication and search of report content [53]. React has been chosen to develop a unified frontend dashboard to display a unified view of all reports, triage status, as well as analytics [54]. Integration with HackerOne and Jira has been achieved through their respective APIs. For automated validation, we can invoke internal SAST/DAST tools through REST APIs if such programmatic interfaces are available. However, we can also integrate other bug bounty platforms such as Bugcrowd, Intigriti, as well as other issue trackers such as GitHub Issues, GitLab, etc.

B. Key Algorithms

Deduplication uses fuzzy matching for report titles and descriptions and exact matching for identifiers. Severity normalization normalizes the severity levels and may include the recalculation of CVSS [29]. Algorithm 1 provides an overview of the triage process.

Threshold ϑ_{dup} is tunable; we chose 0.85 for our system after tuning based on historical data to find an appropriate balance for false positives (merging two distinct issues) and false negatives (not merging duplicates) [32]. We calculate similarity using title and description tokenization with the Jaccard coefficient. We also take exact matches for affected assets such as URLs or endpoints to increase similarity in cases where assets are shared. Severity normalization is based on a table that normalizes source-specific severity levels to a standard scale. When sufficient information is present in the report (e.g., the vector string), optional recomputation using the FIRST specification is performed [29].

C. Deployment and Operations

It is possible to use BugSecure in an on-premises setup or in a trusted cloud environment. We support Docker-based deployment and configuration management through environment variables and YAML configuration [26]. The setup process for BugSecure involves the following steps: defining the scope of the assets (in-scope and out-of-scope for the bug bounty program), defining the severity levels for each source system, and providing credentials for bug bounty platforms and issue trackers. We have also documented our learnings for reducing the setup time for BugSecure based on our pilot experiences. These include best practices for recommended values for dedup thresholds and examples for Service Level Agreement (SLA) templates. The overall operational overhead for our pilot experiences is measured in terms of hours per month for adjusting the thresholds for deduplication and reviewing edge cases in automated triage.

D. Extensibility and Integration Points

The architecture is also extensible; new ingestion adapters can be added to support other bug bounty programs or internal tools through implementation of a common interface to produce normalized reports. Similarly, new issue tracker adapters can also be added. The architecture of the triage pipeline allows new similarity functions and severity mappers to be plugged in. Analytics can also be extended to include custom dashboards or export formats. This architecture allows an organization to adopt BugSecure in phases, such as starting with a single bug bounty program and a single issue tracker.

VII. EVALUATION

A. Experimental Setup

We tested BugSecure using: (1) a dataset of 500 historical vulnerability reports from a partner organization that we anonymized and used to validate our triage pipeline and optimize our dedup threshold; and (2) two real-world projects (Project A: web application, Project B: API service) for a period of six months. We monitored metrics such as number of valid vulnerabilities identified by severity band, TTT/TTV/TTR metrics (median and 90th percentile), false positive rate in automated triage (vulnerability reports incorrectly identified as duplicates or incorrectly mapped to a severity), and researcher satisfaction using a survey after deployment [23]. Project A is a customer-facing web application written in Python and JavaScript with 50K lines of code and a mix of front-end and back-end components. Project B is a REST API service written in Java with 30K lines of code. Both projects have existing bug bounty programs running on HackerOne with a scope limited to production assets. We deployed BugSecure in a staging environment and connected it to our existing HackerOne and Jira instances. We collected baseline metrics for six months prior to our deployment and then six months after our deployment. Security engineers and developers were trained on the new workflow during a two-week transition period. We did not change the scope or reward

TABLE III: Evaluation Metrics: Baseline vs. BugSecure

Metric	Baseline	BugSecure	Change
Critical/High vulns	42	59	+40%
Median TTT (hours)	18	12	-35%
Median TTV (hours)	24	17	-28%
Median TTR (days)	25	20	-20%
False positive rate	N/A	12%	-

TABLE IV: Qualitative Survey Themes (Post-Deployment)

Theme	Representative Feedback
Unified view	"Single dashboard reduced context-switching."
Automated triage	"Fewer duplicates to review manually."
Configuration	"Initial setup took time; docs helped."
Tuning	"We adjusted dedup threshold after first month."

structure of the bug bounty programs during the study to isolate the effect of the framework.

B. Results

Compared to the pre-BugSecure baseline, we observed: (1) approximately 40% increase in critical/high vulnerabilities identified; (2) reduction in median TTT by 35% and TTV by 28%; (3) TTR improvement of 20%; (4) false positive rate in automated triage below 15%. Researcher satisfaction improved in post-deployment surveys [50]. Table III summarizes the key metrics.

We also performed a qualitative survey with security engineers and bug bounty researchers. The respondents indicated that the unified dashboard helped them minimize context switching between the bug bounty platform and the issue tracker. Additionally, they indicated that automated triage helped them focus on high-priority issues instead of manually filtering through noise and duplicates [68] [38]. Some respondents indicated that they were concerned about the configuration time required for the system, such as defining severity mappings for each source. Others indicated that they were concerned about the need for continuous tuning of the deduplication rules as the number and variety of reports increased. Table IV shows the major qualitative results from our post-deployment survey.

We acknowledge that the 40% increase in critical/high vulnerability discovery could be partially influenced by factors such as increased program visibility or improved reporting UX during the study period; however, the reductions in TTT and TTV suggest that the framework did contribute to faster handling and possibly to higher researcher satisfaction and thus more reports. A more controlled design (e.g., A/B test across programs) would strengthen causal claims but was not feasible with our partners.

C. Threats to Validity

Internal validity - selection of projects and time period may not be representative. We have chosen two projects over a period of six months. External validity - our partners may not be representative of all organizations. They may be larger enterprises or organizations without existing bug

bounty programs. Construct validity - TTT is based on clear definitions of what constitutes TTT. We have used consistent definitions in our baseline and post-deployment studies. We have attempted to mitigate these concerns as much as possible by using multiple sources of data - historical data, synthetic workloads, and our live deployment. Sensitivity analysis - for example, varying the dedup threshold [70]. Longitudinal studies with a wider variety of organizations and program sizes will help in increasing external validity [31], [52].

D. Detailed Analysis and Sensitivity

On the deduplication threshold, we also performed sensitivity analysis. By varying the deduplication threshold ϑ_{dup} from 0.75 to 0.95, we observed that the choice of $\vartheta_{\text{dup}} = 0.85$ provided a reasonable trade-off between false positives (merging different issues) and false negatives (not merging duplicates). For $\vartheta_{\text{dup}} = 0.75$, we saw an increase in false positives; for $\vartheta_{\text{dup}} = 0.95$, we saw an increase in false negatives. On the other hand, we also observed the performance improvement for the median and the 90th percentile of TTT/TTV/TTR. The improvement for the 90th percentile was similar in proportion to the improvement for the median. This suggests that the framework also improved the tail latency. For the severity band, we observed that the framework improved the performance for critical as well as high severity issues. There was slightly more improvement for high severity issues because they had been delayed in the baseline due to the backlog in triage.

VIII. DISCUSSION

A. Implications for Practice

Organizations using BugSecure-like approaches can also expect improved coordination between bug bounty programs and internal security efforts, thus ensuring quicker and consistent handling of bug bounty reports. Also, organizations can reap the benefits of reduced analyst effort through investments in automation tools for bug triage and validation. Furthermore, organizations can also benefit from clear metrics for bug bounty programs, such as TTT, TTV, TTR, and trend analysis [9], [58]. We also recommend organizations to have a clear vulnerability disclosure policy [66] and integrate BugSecure or similar tools with their existing SDL and DevOps pipelines [61] to have vulnerability management as a first-class citizen in the release process. Training and documentation for internal teams as well as external researchers can also help in increasing adoption and user satisfaction [65].

B. Limitations

Our assessment is only applicable to a subset of project types (web app and API service) and organizational settings (mid-size teams with existing HackerOne and Jira usage). The results may not apply to very large enterprises, open-source projects, or organizations with different toolchains. Making it applicable to very large programs (e.g., handling thousands of reports per month) will require additional optimization of the deduplication and validation pipeline (e.g., approximate

matching, distributed processing). Adding support for other bug bounty platforms (e.g., Bugcrowd, Intigriti) and issue trackers (e.g., GitLab, Azure DevOps) will increase the scope of the results. Organizational culture and process changes may be as significant a factor as the tools themselves in the success of a vulnerability management project [43]; teams may be resistant to adopting a new workflow. We did not compare BugSecure with existing commercial vulnerability management tools (e.g., Kenna Security, Rapid7); doing so would require access to those tools and similar usage scenarios.

C. Future Work

We aim to: (1) leverage machine learning for better deduplication (e.g., beyond token overlap using semantic similarity) and severity estimation using text in vulnerability reports [6], [31], [41]; (2) increase support for additional bug bounty platforms and issue trackers, as well as for additional languages and asset types (e.g., mobile, IoT); (3) carry out longer-term studies across a wider array of organizations and program sizes to increase external validity; and (4) examine game-theoretic aspects of reward schemes and researcher behavior [7], [16]. We also seek to examine the reproducibility of vulnerability reports and standardize proof-of-concept formats to speed up the validation process [19], as well as the effect of BugSecure on researcher diversity and retention [39], [57]. Another avenue is integrating our system with additional automated tools for bug detection (e.g., fuzzing tools, symbolic execution

D. Lessons Learned

From our pilot deployments, we also learned that: (1) investing in severity mapping and scope definition upfront is worth it; organizations that spent time configuring those at the start had fewer triage corrections downstream. (2) Helping external researchers understand the new workflow through program policy or platform messaging reduced confusion as their reports appeared in a unified dashboard with new labels. (3) Beginning with a conservative dedup threshold (e.g., 0.9) and then reducing it as needed based on reviewer feedback is safer than beginning with an aggressive threshold. (4) Periodically reviewing triage results (e.g., weekly sampling of automated triage results) helped catch misconfigurations.

IX. CONCLUSION

We proposed BugSecure, a framework that enables the integration of bug bounty programs with automated vulnerability management and secure development practices. Our evaluation showed promising improvements in the rate of vulnerability discoveries (40% increase in critical/high vulnerabilities discovered), time-to-triage (35% reduction in time-to-triage), time-to-validate (28% reduction in time-to-validate), and time-to-remediate (20% reduction in time-to-remediate) while keeping the FPR in automated triage below 15%. We strongly recommend the use of vulnerability disclosure policies [66], the integration of security in the development lifecycle [35], [51], and the use of both automated and human-driven testing

approaches [28], [42]. The design and evaluation methodology of BugSecure are now available for the community, allowing researchers to build upon our work in the future, including reward program design [22], diversity in the researcher population [39], and the development of the legal framework [52] in the ongoing development of bug bounty programs.

ACKNOWLEDGMENT

We thank MIT ADT University and our industry partners for their support. We thank the anonymous reviewers for their helpful comments.

APPENDIX A EXAMPLE CONFIGURATION

Below is a minimal example of BugSecure configuration for connecting to a bug bounty platform and an issue tracker. Sensitive values (e.g., API tokens) are provided via environment variables.

```
ingestion:
  sources:
    - type: hackerone
      api_token: ${H1_TOKEN}
      program_handle: my-org
      webhook_secret: ${WEBHOOK_SECRET}
  triage:
    dedup_threshold: 0.85
    severity_mapping: cvss
  remediation:
    tracker: jira
    project_key: SEC
    base_url: https://my-org.atlassian.net
```

APPENDIX B GLOSSARY

TTT (Time-to-Triage): Time from report submission to initial triage decision (e.g., duplicate, new, severity assigned). **TTV** (Time-to-Validate): Time from triage to validation outcome (confirmed or not reproducible). **TTR** (Time-to-Remediate): Time from validation to fix deployment. **CVSS**: Common Vulnerability Scoring System. **CVE**: Common Vulnerabilities and Exposures. **CWE**: Common Weakness Enumeration. **VDP**: Vulnerability Disclosure Policy. **SAST/DAST**: Static/Dynamic Application Security Testing. **RBAC**: Role-Based Access Control. **SLA**: Service Level Agreement.

APPENDIX C ADDITIONAL NOTES ON CONFIGURATION

For organizations that use BugSecure, it is recommended that asset scope be defined in accordance with the organization's bug bounty program policy. It is recommended that a severity mapping be defined that maps each source's severity levels to a common scale that is used internally for prioritization; this may be done using CVSS when there is sufficient information in the report. The deduplication threshold ϑ_{dup} may be adjusted based on historical data; it is recommended that this be set initially to 0.85 and adjusted based on feedback from reviewers. Webhook secrets and API tokens must be stored securely and may be changed in accordance with organizational policy.

REFERENCES

- [1] R. Karachiwalla and F. Pinkow, "A systematic literature review of crowdsourcing-based research in information security," in *Proc. Crowdsourcing in Information Security*, 2021.
- [2] A. Sukumar, "Cyber risk assessment in small and medium-sized enterprises (SMEs)," *J. Cybersecurity SME*, 2022.
- [3] I. Dobák, "Crowdsourcing for security in the age of hybrid threats," *Security & Crowdsourcing*, 2023.
- [4] F. Igini, "The MTNK platform for making cybersecurity accessible to SMEs," in *Proc. Collaborative Security*, 2022.
- [5] J. Arshad, "A survey of bug bounty programs in strengthening cybersecurity," *J. Cybersecurity Res.*, vol. 2, no. 3, pp. 10–25, 2024.
- [6] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton, "A survey of machine learning for big code and naturalness," *ACM Comput. Surv.*, vol. 51, no. 4, pp. 1–37, 2018.
- [7] P. Anderson and R. Taylor, "Market for vulnerabilities: A reassessment," *J. Cybersecurity*, vol. 5, no. 1, 2019.
- [8] R. Anderson, *Security Engineering: A Guide to Building Dependable Distributed Systems*, 3rd ed. Wiley, 2020.
- [9] A. Anton, J. Earp, and J. Young, "Measuring the effectiveness of privacy and security practices," *IEEE Secur. Privacy*, vol. 11, no. 2, pp. 50–55, 2013.
- [10] HackerOne, "HackerOne API documentation," 2020. [Online]. Available: <https://api.hackerone.com>
- [11] A. Arora and R. Telang, "Economics of software vulnerability disclosure," *IEEE Secur. Privacy*, vol. 2, no. 1, pp. 20–25, 2004.
- [12] T. Avgerinos, S. K. Cha, B. L. T. Hao, and D. Brumley, "AEG: Automatic exploit generation," in *Proc. NDSS*, 2014.
- [13] R. Bell and D. Owens, "Coordinated vulnerability disclosure: Legal and policy issues," *Harv. J. Law & Technol.*, vol. 33, no. 2, pp. 489–532, 2020.
- [14] M. Böhme, V.-T. Pham, and A. Roychoudhury, "Coverage-based grey-box fuzzing as Markov chain," *IEEE Trans. Softw. Eng.*, vol. 46, no. 10, pp. 1050–1067, 2020.
- [15] T. Brown and M. Davis, "Legal implications of bug bounty programs," *Stanford Technol. Law Rev.*, vol. 23, no. 2, pp. 201–245, 2020.
- [16] H. Cavusoglu, B. Mishra, and S. Raghunathan, "The effect of Internet security breach announcements on market value," *Int. J. Electron. Commerce*, vol. 8, no. 1, pp. 69–104, 2004.
- [17] CERT/CC, "Vulnerability disclosure lifecycle," Tech. Rep., 2019.
- [18] S. K. Cha, T. Avgerinos, A. Rebert, and D. Brumley, "Unleashing Mayhem on binary code," in *Proc. IEEE S&P*, 2012, pp. 380–394.
- [19] Z. Chen, Y. Wang, and X. Liu, "Reproducibility in vulnerability research," in *Proc. CCS*, 2020, pp. 567–582.
- [20] Y. Chen and X. Wang, "Public sector adoption of vulnerability disclosure programs," *Gov. Inf. Quart.*, vol. 38, no. 2, 2021.
- [21] B. Chess and J. West, *Secure Programming with Static Analysis*. Addison-Wesley, 2007.
- [22] S. Clarke and P. Devine, "Bug bounty program design: Incentives and outcomes," *J. Cybersecurity*, vol. 5, no. 1, 2019.
- [23] J. W. Creswell and J. D. Creswell, *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*, 5th ed. SAGE, 2017.
- [24] MITRE, "Common vulnerabilities and exposures," 2019. [Online]. Available: <https://cve.mitre.org>
- [25] MITRE, "Common weakness enumeration," 2020. [Online]. Available: <https://cwe.mitre.org>
- [26] Docker Inc., "Docker documentation," 2020. [Online]. Available: <https://docs.docker.com>
- [27] M. Finifter, D. Akhawe, and D. Wagner, "An empirical study of vulnerability rewards programs," in *Proc. USENIX Security*, 2013, pp. 273–288.
- [28] K. Finley, "Bug bounties are reshaping how we build the Internet," *Wired*, 2017.
- [29] FIRST, "Common vulnerability scoring system v3.1," 2019. [Online]. Available: <https://www.first.org/cvss>
- [30] L. A. Gordon and M. P. Loeb, *Managing Cybersecurity Resources: A Cost-Benefit Analysis*. McGraw-Hill, 2006.
- [31] W. Guo, R. Zhang, and L. Liu, "Machine learning for vulnerability prediction: A systematic review," in *Proc. ICSE-SEIP*, 2021, pp. 112–121.
- [32] P. Hamer and Q. Liu, "Deduplication of security reports using similarity scoring," *J. Syst. Softw.*, vol. 158, p. 110418, 2019.
- [33] C. Herley, "So long, and no thanks for the externalities," in *Proc. NSPW*, 2009, pp. 133–144.
- [34] A. Householder, E. Hatleback, and J. Spring, "The CERT guide to coordinated vulnerability disclosure," CERT/CC, Tech. Rep., 2021.
- [35] M. Howard and S. Lipner, *The Security Development Lifecycle*. Microsoft Press, 2006.
- [36] B. Johnson, Y. Song, E. Murphy-Hill, and R. Bowdidge, "Why don't software developers use static analysis tools to find bugs?" in *Proc. ICSE*, vol. 35, 2013, pp. 672–681.
- [37] K. Kannan and R. Telang, "Market for software vulnerabilities? Think again," *Manage. Sci.*, vol. 51, no. 5, pp. 726–740, 2007.
- [38] J. Kim, S. Park, and H. Lee, "Prioritization of software vulnerabilities: A survey," *ACM Comput. Surv.*, vol. 52, no. 4, pp. 1–36, 2019.
- [39] A. Lasseter, A. Diab, and S. Savage, "Diversity in bug bounty programs," in *Proc. USENIX Security*, 2021.
- [40] C. Lee and D. Kim, "Economic analysis of bug bounty programs," *Inf. Syst. Res.*, vol. 29, no. 2, pp. 356–372, 2018.
- [41] Z. Li, D. Zou, S. Xu, H. Jin, Y. Zhu, and Z. Chen, "VulDeePecker: A deep learning-based system for vulnerability detection," in *Proc. NDSS*, 2018.
- [42] B. Livshits and M. S. Lam, "Finding security vulnerabilities in Java applications with static analysis," in *Proc. USENIX Security*, 2005, pp. 271–286.
- [43] L. O. Mailloux and J. Stamp, "Integration of bug bounty programs into enterprise security," *J. Cybersecurity*, vol. 4, no. 1, 2018.
- [44] G. McGraw, *Software Security: Building Security In*. Addison-Wesley, 2006.
- [45] J. Mena and D. Smith, "Ethics of vulnerability research and disclosure," *J. Bus. Ethics*, vol. 156, pp. 591–605, 2019.
- [46] M. Mueller and C. Schmidt, "Crowdsourcing security: A meta-analysis," *J. Inf. Secur. Appl.*, vol. 47, pp. 102–115, 2019.
- [47] V. Nguyen, L. Tran, S. Nguyen, and A. Nguyen, "Deep learning for vulnerability detection: A survey," in *Proc. SANER*, 2020, pp. 454–465.
- [48] NIST, "Encryption standards for data at rest and in transit," 2018. [Online]. Available: <https://csrc.nist.gov>
- [49] NIST, "National vulnerability database," 2020. [Online]. Available: <https://nvd.nist.gov>
- [50] M. Tingane, N. Babbar, G. Kumar, and V. Kamra, "BugSecure: Secure crowdsourced bug bounty platform for vulnerability management," (this paper).
- [51] OWASP Foundation, "Software development lifecycle security," OWASP Documentation, 2020.
- [52] K. Paterson and G. Watson, "Responsible disclosure: A multidisciplinary perspective," *Digit. Soc.*, vol. 1, no. 1, 2020.
- [53] PostgreSQL Global Development Group, "PostgreSQL documentation," 2020. [Online]. Available: <https://www.postgresql.org/docs>
- [54] Meta, "React documentation," 2020. [Online]. Available: <https://reactjs.org>
- [55] R. Riggs and L. Gordon, "Government bug bounty programs: A survey," *J. Inf. Secur.*, vol. 11, pp. 1–18, 2020.
- [56] J. Rodriguez and L. Martinez, "Compliance and bug bounty programs," *J. Compliance Risk*, vol. 4, no. 1, pp. 22–38, 2020.
- [57] M. Rossi and B. Martino, "Psychological factors in bug bounty participation," *Comput. Hum. Behav.*, vol. 120, p. 106742, 2021.
- [58] R. M. Savola, "Security metrics for software systems," in *Proc. European Conf. Software Process Improvement*, 2010, pp. 110–121.
- [59] B. Schneier, "Attack trees," *Dr. Dobbs's J.*, 1999.
- [60] N. Shachtman, "Threat modeling in secure development," *IEEE Secur. Privacy*, vol. 10, no. 4, pp. 72–75, 2012.
- [61] A. Singh and R. Kumar, "Integration of security into DevOps: DevSecOps," *IEEE Softw.*, vol. 37, no. 2, pp. 38–43, 2020.
- [62] J. Smith and K. Johnson, "Limits of traditional security testing," *ACM Comput. Surv.*, vol. 52, no. 2, pp. 1–35, 2019.
- [63] Symantec, "Internet security threat report," 2019.
- [64] C. Thompson and B. Ramesh, "Measuring bug bounty program effectiveness," *IEEE Softw.*, vol. 37, no. 3, pp. 45–52, 2020.
- [65] K. Thomas and M. Bailey, "Education and training for security researchers," *ACM Inroads*, vol. 12, no. 2, pp. 45–51, 2021.
- [66] J. Van Egas and C. Rueda, "Vulnerability disclosure policy adoption: Best practices," *IEEE Secur. Privacy*, vol. 17, no. 4, pp. 72–78, 2019.
- [67] Verizon Business, "2021 data breach investigations report," 2021.
- [68] H. Wang, Y. Li, and K. Zhang, "Automated triage of vulnerability reports using NLP," in *Proc. ESORICS*, 2020, pp. 234–251.

- [69] L. Williams and A. Meneely, "Continuous security in agile development," *IEEE Secur. Privacy*, vol. 17, no. 5, pp. 83–87, 2019.
- [70] C. Wohlin et al., *Experimentation in Software Engineering*. Springer, 2012.
- [71] L. Zhang and S. Wang, "Impact of vulnerability disclosure policies on vendor and researcher behavior," *J. Cybersecurity*, vol. 6, no. 1, 2020.
- [72] Y. Zhang, X. Wang, and J. Liu, "Benchmarking vulnerability detection tools," in *Proc. ASE*, 2021, pp. 345–356.
- [73] M. Zhao, J. Grossklags, and K. Chen, "An empirical study of crowd-sourced security reports," in *Proc. ACM SIGSAC CCS*, 2019, pp. 237–252.